RLE (Run Length Encoding) to demonstrate basic compression

Karan Kadam
9th July 2011

http://synapse.wordpress.com

Overview

RLE (Run Length Encoding) is the most basic kind of data compression. This paper is about understanding this encoding technique and then implementing it using ANSI C. When we think of compression, data redundancy comes to mind. RLE is getting rid of this redundancy by organizing the source symbols in a specific manner. To quote Wikipedia on the subject –

"Run-length encoding (RLE) is a very simple form of data compression in which runs of data (that is, sequences in which the same data value occurs in many consecutive data elements) are stored as a single data value and count, rather than as the original run. This is most useful on data that contains many such runs: for example, simple graphic images such as icons, line drawings, and animations. It is not useful with files that don't have many runs as it could greatly increase the file size."

The paragraph above clearly states that this type of encoding is most effective when multiple runs of data (consecutive redundant data) are found in the original source. This technique is not suitable for randomized data with less repetition and redundancy.

Basic RLE

The most basic RLE can be achieved using the following technique. Although it has its limitations, it is presented here merely for the sake of clarity. This is the most simple and basic type of RLE.

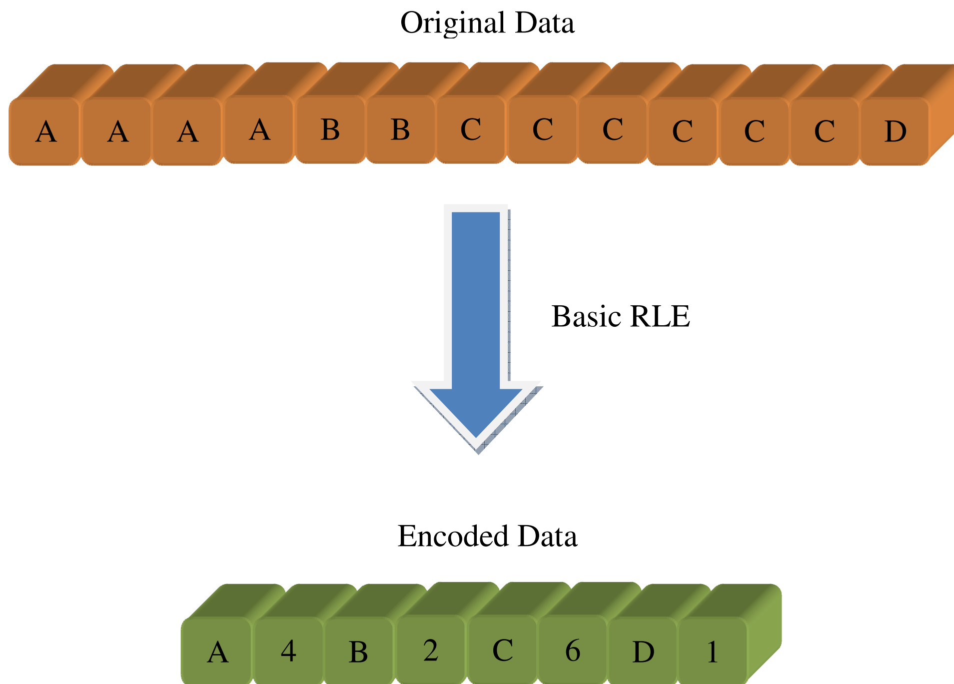

Figure – A basic RLE scheme

It can be seen from the figure above that that the overall size of the data has reduced as the redundant symbols of our data have been re-organized in a much better manner. If we consider each block to be 1 byte, we have reduced the size of our data from 13 bytes to 8 bytes which is a ~1:1.8 compression ratio.

In the scheme above, the numbers 4, 2, 6 and 1 correspond to **"run lengths"** and hence this encoding scheme gets its name.

However, the above encoding scheme has a small problem. How do we know if the symbol which follows a character symbol is a numerical symbol or a run length? In other words, our source data could contain numerical symbols as well. This will also cause problems when we try to decode the data. Also, another disadvantage of this scheme is that for a single character symbol say "D" (1 byte), we would have to store it as "D,1" (2 bytes).

Hence, we make a small change to the above scheme to overcome these problems.

Improved RLE

A small change in the way data is organized in Basic RLE will be incorporated in order to overcome the difficulties faced in the above mentioned scheme. This modified scheme is an "improved" version of Basic RLE and can be illustrated by the figure below –

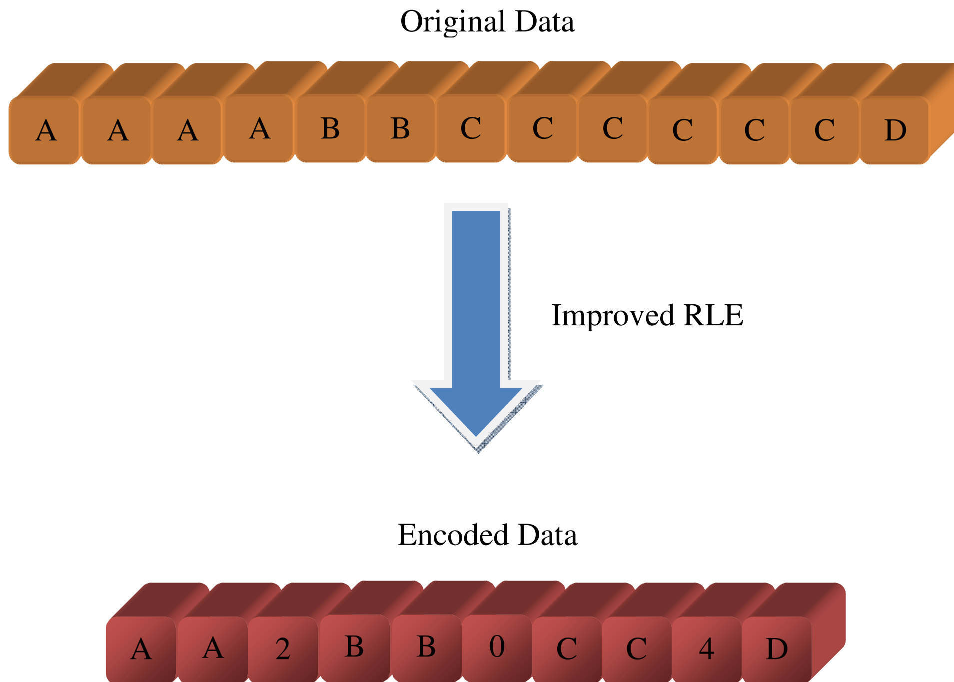

Figure – A basic RLE scheme

The Improved RLE scheme illustrated above solves the problem of not knowing whether we are dealing with a numerical symbol or a run length. While encoding, we store symbols using the formula NN(X-2), where 'X' is the total no. of occurrences of the symbol and 'N' is the symbol. "X-2" will give us the run length.

A is stored simply as A
AA is stored as AA(2-2) = AA0
AAA is stored as AA(3-2) = AA1
AAAA is stored as AA(4-2) = AA2 and so on.

While decoding, whenever we encounter the same symbols twice in a row, we know that the next symbol after this will be the run length. For single symbols like "D" above we do not store a run length.

We have now compressed (encoded) 13 bytes to 10 bytes. Although the compression ratio does not seem like much, remember that this encoding is most beneficial for data that contains numerous runs and will show a high compression ratio when the run length is high. Let us now look at how we will incorporate this scheme using a high level language such as ANSI C.

Improved RLE Encoding

The simplified algorithm for the encoding scheme can be illustrated by the steps below –

1. Start.
2. Assign previous symbol as invalid value such as “EOF”.
3. Read one symbol from source (next symbol).
4. If next symbol != previous symbol, assign previous symbol = next symbol and read next symbol.
5. While (next symbol == previous symbol), increase run length. Note: Run length = 0 if match count = 2 (AA0), Run length = 1 if match count = 3 (AA1) and so on.
6. Loop until next symbol = EOF (end of file).
7. Stop.

Improved RLE Decoding

The simplified algorithm for the decoding scheme can be illustrated by the steps below –

1. Start.
2. Assign previous symbol as invalid value.
3. Read one symbol from source (next symbol).
4. If next symbol != previous symbol, assign previous symbol = next symbol and read next symbol.
5. If (next symbol == previous symbol), read next symbol.
6. Next symbol is the run length.
7. Print previous symbols equal to run length count.
8. Loop until symbol = EOF (end of file).
9. Stop.

Improved RLE implementation in ANSI C

The above algorithm has been implemented by me in plain C. The following code has been tested using a standard C++ compiler. To really see the benefits of RLE compression, use data which has multiple run lengths. For regular data you may not see much (if any) benefit.

```
/*
Description: RLE encoding to demonstrate basic compression
Author: Karan Kadam
Date: 9th June 2011
Version: v0.1

Overview:
aaaaaaaabbbcddddddddddddd(24bytes) => aa6bb1cc0dd10(13bytes)

Basic idea taken from:
https://secure.wikimedia.org/wikipedia/en/wiki/Run-length_encoding

Authors Blog:
http://synapse.wordpress.com
*/

/*Includes*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

/*Defines*/
#define FILE_PATH 512
#define OPTION 1
#define INFILE 2
#define OUTFILE 3

/*
EncodeRLE(): Function to encode characters so as to remove redundancy
szInfile: File to be encoded/compressed
szOutfile: Encoded(compressed) file

Retval:
1: Success
0: Failure

Algorithm used (example):
aaaaaaaabbbcddddddddddddd(24bytes) => aa6bb1cc0dd10(13bytes)
*/
int EncodeRLE(char* szInfile, char* szOutfile)
{
	/*File handles*/
	FILE* fpIn = NULL;
	FILE* fpOut = NULL;

	/*Previous and current characters*/
	int prevChar = EOF;
	int nextChar = 0;

	/*Run length - No. of repeats*/
	int count = 0;
```

```
	/*Open input file for reading*/
	if(NULL == (fpIn=fopen(szInfile,"rb")))
	{
		fprintf(stderr,"Error opening input file:
%s\n",szInfile);
		return 0;
	}

	/*Open output file for writing*/
	if(NULL == (fpOut=fopen(szOutfile,"wb")))
	{
		fprintf(stderr,"Error opening output file:
%s\n",szOutfile);
		return 0;
	}

	/*
	Get chars one by one as long as we reach end of file.
	If previous and next characters match, we have a repeat.
	If multiple characters match, we calculate the runlength.
	*/
	while(EOF != (nextChar = fgetc(fpIn)))
	{
		/*Write one char to file*/
		fputc(nextChar,fpOut);

		/*Repeat found!*/
		if(prevChar == nextChar)
		{
			/*Current run length is 0*/
			count = 0;

			while(EOF != (nextChar = fgetc(fpIn)))
			{
				/*We need a run length count now*/
				if(prevChar == nextChar)
				{
					count++;
				}
				else
				{
					/*No more matches. Write char and run
length(count)*/
					fputc(count,fpOut);
					fputc(nextChar,fpOut);
					prevChar = nextChar;
					break;
				}
			}/*End inner while*/
		}
		else
		{
			prevChar = nextChar;
		}

	}/*End outer while*/

	/*Write count for final character*/
	fputc(count,fpOut);

	/*Clean up*/
```

```
	fclose(fpIn);
	fclose(fpOut);

	return 1;
}

/*
DecodeRLE(): Function to decode a RLE encoded file
szInfile: File to be decoded/decompressed
szOutfile: Decoded(decompressed) file

Retval:
1: Success
0: Failure

Algorithm used (example):
aa6bb1cc0dd10(13bytes) => aaaaaaaabbbcdddddddddddd(24bytes)
*/
int DecodeRLE(char* szInfile, char* szOutfile)
{
	/*File handles*/
	FILE* fpIn = NULL;
	FILE* fpOut = NULL;

	/*Loop counter*/
	int nCnt;

	/*Previous and current characters*/
	int prevChar = EOF;
	int nextChar = 0;

	/*Run length - No. of repeats*/
	int count = 0;

	/*Open input file for reading*/
	if(NULL == (fpIn=fopen(szInfile,"rb")))
	{
		fprintf(stderr,"Error opening input file:
%s\n",szInfile);
		return 0;
	}

	/*Open output file for writing*/
	if(NULL == (fpOut=fopen(szOutfile,"wb")))
	{
		fprintf(stderr,"Error opening output file:
%s\n",szOutfile);
		return 0;
	}

	while(EOF != (nextChar = fgetc(fpIn)))
	{
		fputc(nextChar,fpOut);

		/*We have a repeat*/
		if(prevChar == nextChar)
		{
			/*Get the repeat count (Run length)*/
			nextChar = fgetc(fpIn);

			/*Write characters as per run length*/
```

```
			for(nCnt = 0; nCnt < (int)nextChar ; nCnt++)
			{
				fputc(prevChar,fpOut);
			}
		}
		else
		{
			prevChar = nextChar;
		}

	}/*end outer while*/

	/*Clean up*/
	fclose(fpIn);
	fclose(fpOut);

	return 1;
}

/*
main(): Main body of program
argc: Argument counter
argv: Argument vector
*/
int main(int argc, char** argv)
{
	char szInfile[FILE_PATH];
	char szOutfile[FILE_PATH];

	/*Compress mode(Option i.e argv[2])*/
	int nCompressMode = 1;

	/*Banner*/
	fprintf(stderr,"RLE demonstration\n");
	fprintf(stderr,"Karan Kadam - 9th July 2011\n");
	fprintf(stderr,"http://synapse.wordpress.com\n\n");

	memset(szInfile,0,FILE_PATH);
	memset(szOutfile,0,FILE_PATH);

	/*Display correct usage help*/
	if(4 != argc)
	{
		fprintf(stderr,"Incorrect usage\n");
		fprintf(stderr,"Please use RLETry.exe <option> <infile>
<outfile>\n");
		fprintf(stderr,"Options -\n");
		fprintf(stderr,"-e Encode \n");
		fprintf(stderr,"-d Decode\n");

		exit(1);
	}

	/*Get input filename*/
	strcpy(szInfile,argv[INFILE]);

	/*Get output filename*/
	strcpy(szOutfile,argv[OUTFILE]);

	/*Get option*/
	((strcmp(argv[OPTION],"-E")) || (strcmp(argv[OPTION],"-e")))?
```

```
		(nCompressMode = 0):(nCompressMode = 1);

	if(nCompressMode)
	{
		/*Perform encode operation*/
		if(EncodeRLE(szInfile,szOutfile))
		{
			fprintf(stdout,"Encoded file succesfully.\n");
		}
		else
		{
			fprintf(stdout,"Error while encoding file!\n");
			return 1;
		}
	}
	else
	{
		/*Perform decode operation*/
		if(DecodeRLE(szInfile,szOutfile))
		{
			fprintf(stdout,"Decoded file succesfully.\n");
		}
		else
		{
			fprintf(stdout,"Error while decoding file!\n");
			return 1;
		}
	}

	return 0;
}
```

About Me

I have been interested in Computers especially Linux, Reverse Engineering and Network Security for the last 10 years. A Software Developer by profession, I love playing video games and enjoy clicking photographs occasionally. To view some photos I have clicked visit http://karankadam.blogspot.com. For other information like this article visit my technical blog at http://synapse.wordpress.com

If you have any comments or suggestions for this paper please contact me at karankadam at gmail dot com. Thanks for reading this article.

References

http://en.wikipedia.org/wiki/Run-length_encoding
http://michael.dipperstein.com/rle/index.html
http://www.arturocampos.com/ac_rle.html